Reverse Debugging of Kernel Failures in Deployed Systems

Xinyang Ge, Ben Niu and Weidong Cui
Microsoft Research

USENIX Annual Technical Conference, 2020
What happened before the crash?

Working on updates
11% complete
Don’t turn off your computer

:(

Your PC ran into a problem and needs to restart. We’re just collecting some error info, and then we’ll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/shutdown

You can restart by pressing Ctrl+Alt+Delete and choosing Restart.
REPT: Reverse Execution with Processor Trace
**REPT: Reverse Execution with Processor Trace**

- A practical reverse debugging solution for user-mode failures [OSDI’18]

- Online hardware tracing (e.g., Intel Processor Trace)
  - Log the control flow with timestamps
  - Low runtime overhead (1-5%)
  - *No data!*

- Offline binary analysis
  - Recovers data flow from the control flow

*How to make REPT support the kernel?*
How REPT works?
How REPT works?
How REPT works?
How REPT works?
How REPT works?

rax = 3, rbx = 1

add rax, rbx
rax = ?, rbx = ?

BOOM!
How REPT works?

```
add rax,rbx
rax=2,rbx=1
rax=3,rbx=1
```
Can we simply inverse the tracing?
Can we simply inverse the tracing?

• There are too many processes/threads on a system
  • High memory overhead for tracing

• Hardware events must be emulated in addition to CPU instructions
  • Interrupts
  • Exceptions
  • System calls
Here comes Kernel REPT...
context switch

... is irreversible, and we log it in software.
syscalls
interrupts/exceptions

USER
KERNEL
Different events can have different architectural effects.
That’s it?
Automated Analyses

• A common bug pattern: missing undo operations
  • EnterCriticalRegion vs LeaveCriticalRegion

• Root-Cause Analysis
  • Scan the kernel execution trace to find missing undo operations

• Proactive Bug Detector
  • Sanitize the kernel execution based on specified invariants
  • 17 new bugs found and fixed!
Demo
Conclusion

• Debugging production kernel failures is hard

• REPT now supports the reverse debugging of the kernel
  • Per-core control flow tracing in hardware
  • Context switch logging in software
  • Recovers data flow via CPU instruction and hardware event emulation

• REPT enables automated analysis beyond reverse debugging
  • Root-cause analysis
  • Sanitizing analysis